L5: Entry 4 of 4 File: USPT Oct 5, 1999

DOCUMENT-IDENTIFIER: US 5963933 A
TITLE: Efficient implementation of full outer join and anti-join

Abstract Text (1):
Method for specifying SQL "full outer joins" which enables the use of arbitrary join conditions in specifying the query. This is enabled by equating a full outer join with a left outer join unioned with a right outer join less the matched tuples from the right outer join. A number of embodiments further present improvements in execution speed. One such improvement utilizes as a second operand of the union query a novel operator, the "ANTI-JOIN". The anti-join is implemented as a right outer join coupled with an ISNULL predicate.

Brief Summary Text (3):
This invention relates to query mechanisms for relational database management systems. More particularly, the present invention relates to a novel method of evaluating SQL "full outer joins", which methodology enables the use of arbitrary join conditions in specifying the query.

Brief Summary Text (14):
This query returns all the matching pairs of T1 and T2 tuples based on the join condition "q1.c1>=q2.c2", just as in a regular join. The query also returns rows of T1 which do not match any T2 rows based on the join condition. In this example, T1 is called the tuple-preserving operand, while T2 is called the null-producing operand, because T1 tuples will be preserved when there is no match in T2.

Brief Summary Text (19):
This query returns all the matching pairs of T1 and T2 tuples based on the join condition "q1.c1>=q2.c2". The query also returns rows of T1 which do not match any T2 rows based on the join condition as well as rows of T2 which do not match any T1 rows.

Brief Summary Text (22):
Many database management systems support left and right outer joins. Some systems further support full outer joins, but they impose the strict limitation that only the conjunction of equality predicates is allowed as the join condition. This limitation is due to the fact that these systems implement the full outer join using a modified merge join algorithm which, prior to the making of the present invention, has inherently been restricted to the use of equality predicates.

Brief Summary Text (24):
In general, each of these references regarding outer join implementation shares common theme: the strategies taught therein require special runtime operators, which somehow keep track of which tuples must be preserved. In the prior art this has typically been done by creating and maintaining some list of tuple identifiers, or TIDs, for those tuples which must be preserved. When the outer join condition involves only equality predicates in the conjunctive normal form, it is possible to implement the full outer join via a modified merge join algorithm without keeping lists of preserved tuple identifiers. However, a modified merge join, according the

prior art, cannot be used for arbitrary join conditions, including but not necessarily limited to the inequality predicate and disjunction, as specified in IPSO-ANSI92. Moreover, the creation and maintenance of TID lists introduces an unwanted and now unnecessary element of computational overhead, thereby inducing system performance problems in any system which relies on such lists.

Brief Summary Text (25):
Join conditions are well defined in IPSO-ANSI92. Typically, join conditions defining the relationship between tuples are logically selectable from the following set of predicates:

Brief Summary Text (34):
A plurality of these conjunction predicates can also be logically joined by the logical operators AND, OR, and NOT. Accordingly, as used herein, the term "arbitrary join conditions" defines any of these join conditions, or a plurality of join conditions joined by one of the defined logical operators. This precisely defined term defines over prior art full outer join methodologies, which support only the equality predicate.

Brief Summary Text (35):
Finally, the computation of large queries implementing a full outer join can be resource intensive. This problem is often exacerbated by increasingly complex query structures where the union of a plurality of outer joins repeatedly returns a number of instances of a matching pair. Any solution to the problem of using a modified merge join to implement a full outer join and which further enables the use of arbitrary join conditions should, to the greatest extent possible, minimize the impact of the query on system resources by only returning the first instance of a matching pair.

Brief Summary Text (36):
Accordingly, there is a clearly-felt need in the art for a methodology which enables the use of a modified merge join to implement a full outer join which enables the use of arbitrary join conditions other than the equality predicate. There is a further need that this query methodology return only a first instance of a matching pair in response to a join condition, thereby enabling the methodology to be resource efficient. These unresolved problems are clearly felt in the art and are solved by this invention in the novel manner described below.

Brief Summary Text (38):
This invention devises, for the first time, a mechanism which enables the implementation of the full outer join with any join condition, including but not necessarily limited to the inequality predicate and disjunction. This is enabled in one embodiment of the present invention by equating, in a novel manner, a full outer join with a left outer join unioned with a right outer join less the matched tuples from the right outer join. To implement this embodiment, a full outer join query is programmatically rewritten as a union of a left outer join and a select query with a "not-exists" subquery. Indeed, the present invention provides, for the first time, a methodology for properly unioning a left outer join and a right outer join whereby the result is a properly formed full outer join producing a correct answer set.

Brief Summary Text (40):
Where an outer join operand does not have any non-nullable column, for instance as when the operand is a derived table, a further improved embodiment adds a column of constants to achieve the same effect as the previous embodiment. For example, suppose both tables DT1 and DT2 are not base tables: i.e., their row identifiers are not readily available, and all columns are nullable. By adding a column, or key, of constant value 1, or indeed of any other constant, in the operand of the right outer join, it can be assured that the column value is null for the preserved tuples of DT2. Hence, it is always possible to select those DT2 rows that do not

have any matching rows in DT1.

Brief Summary Text (42):
It is emphasized that the principles of the present invention achieve the several features and advantages hereof without any restriction on the join condition for the full outer join. This is true because there is no restriction on the join condition for left and right outer joins.

Detailed Description Text (15):
A first preferred embodiment of the present invention devises, for the first time, a mechanism which enables a query implementing a full outer join with any join conditions, including but not necessarily limited to the inequality predicate and disjunction. This is enabled by equating, in a novel manner, a full outer join with a left outer join unioned with a right outer join less the matched tuples from the right outer join. Referring to FIG. 4, a flow diagram of this embodiment is shown. At 401, the routine is started, and at 402, the outer join to be performed is identified. At 403, the full outer join is programmatically transformed into a union of a left outer join and a right outer join, minus the matched tuples from the right outer join. The rewritten SQL statement which implements the transformed full outer join is provided to the RDBMS' optimizer (not shown) at 404, and program execution continues at 405. Consider a full outer join with join condition P(T1, T2):

Detailed Description Text (18):
It is recognized that transforming a full outer join into a union query with a not-exists subquery may not provide optimally efficient computational performance. This is due to the fact that many subqueries are inherently slow to execute, especially in the MPP shared-nothing environment. Accordingly, the second preferred embodiment of the present invention presents a novel and computationally efficient full outer join with an arbitrary join condition capability.

Detailed Description Text (19):
Having reference to FIG. 5, this embodiment of the present invention also transforms the full join query into a union query of left outer join and right outer join, at 501-503. However, in order to ensure that the query returns the correct answer, for the right outer join all the matched rows are filtered out from 504-506. To indicate that the output from the right outer join is a matched row, a non-nullable column of the null-producing operand is added to the output of the right outer join at 504. The non-nullable column includes a primary key (key), and a row identifier (rid) or tuple identifier (tid). The value of these columns is set to null at 505 when there is not a match on the tuple-preserving operand by the definition of the outer join. By applying the "IS NULL" predicate after the right outer join at 506, all matched rows are removed from the answer set. The rewritten SQL statement which implements the transformed full outer join is provided to the RDBMS' optimizer (not shown) at 507, and program execution continues at 508.

Detailed Description Text (22):
The preceding approach is acceptable given a non-nullable column. FIG. 6 details another preferred embodiment useful where an outer join operand does not have any non-nulable column, for instance when the operand is a derived table, a column of constants could be added to achieve the same effect. For example, suppose both DT1 and DT2 are not base tables: i.e., their row identifiers were not readily avialable, and all columns were nullable. A full outer join for such a condition could then be rewritten as:

Detailed Description Text (24):
Having continued reference to FIG. 6, this embodiment of the present invention also transforms the full join query into a union query of left outer join and right join, at 601-603. However, in order to ensure that the query returns the correct answer, for the right outer join all the matched rows are filtered out from 604-

606. By adding a column of constant 1 at 604, or any other constant, in the null-producing operand of the right outer join, it is assured that the column value is null for the preserved tuples of DT2 (q4) at 605. By applying the "IS NULL" predicate after the right outer join at 606, all matched rows are removed from the answer set. The rewritten SQL statement which implements the transformed full outer join is provided to the RDBMS' optimizer (not shown) at 607, and program execution continues at 608. Hence, those DT2 rows that do not have any matching rows in DT1 are always selectable. This then discloses the third preferred embodiment of the present invention.

Detailed Description Text (27):
Given tables T1 and T2, and ajoin condition P(T1, T2), the anti-join (T1, T2, P(T1, T2)) returns all T1 tuples that do not match with T2 tuples using join condition P (T1, T2).

Detailed Description Text (28):
Having continued reference to FIG. 7, in this embodiment after the full outer join is identified at 702, it is transformed as the union of a left outer join and an anti-join at 703. The rewritten SQL statement which implements the transformed full outer join is provided to the RDBMS' optimizer (not shown) at 704, and program execution continues at 705.

Detailed Description Text (34):
To implement a full outer join using anti-join, consider a full outer join discussed above with join condition P(T1, T2):

Detailed Description Text (37):
It should be emphasized that because the approach taught herein for implementing a full outer join utilizes the properties of the left/right outer join, there is no restriction on the join condition for the full outer join because there is no restriction on the join condition for the left/right outer join which enables it.

CLAIMS:

1. In a relational database management system utilizing a data processor for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of tuples having one or more columns, wherein the data contained in the relational database management system is retrievable by means of query language queries to the database management system, a data processor implemented method for returning an answer set from the data in the tables responsive to a query to the database management system, the answer set requiring the matching of tuples from the first table and the second table, the method comprising the step of joining the tuples of the first and second tables using a full outer join with a join condition, the join condition being free of restrictions as to the type of join condition specified, the step of joining including unioning a left outer join with a right outer join less the matched tuples from the right outer join.

2. The method of claim 1 wherein the join condition is from the set of comparison operators consisting of: IS NULL; LIKE; EQUALS; DOES NOT EQUAL; IS GREATER THAN; IS LESS THAN; IS GREATER THAN OR EQUAL TO; and IS LESS THAN OR EQUAL TO.

4. In a relational database management system utilizing a data processor for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of tuples having at least one column, wherein the data contained in the relational database management system is retrievable by means of query language queries to the database management system, a data processor implemented method for returning an answer set from the data in the tables responsive to a query to the database management system, the answer set requiring the matching of tuples from the first

table and the second table, the method comprising the step of joining the tuples of the first and second tables using a full outer join with any join condition specifiable in the query language, the step of joining including unioning a left outer join with a right outer join less the matched tuples from the right outer join.

5. The method of claim 4 wherein the join condition is from the set of comparison operators consisting of: IS NULL; LIKE; EQUALS; DOES NOT EQUAL; IS GREATER THAN; IS LESS THAN; IS GREATER THAN OR EQUAL TO; and IS LESS THAN OR EQUAL TO.

7. In a relational database management system utilizing a data processor for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of tuples having at least one column, wherein the data contained in the relational database management system is retrievable by means of query language queries to the database management system, a data processor implemented method for returning an answer set from the data in the tables responsive to a query to the database management system, the answer set requiring the matching of tuples from the first table and the second table, the method comprising the steps of:

for a full outer join condition, receiving a full outer join query of the tuples of the first operand and the tuples of the second operand, with a join condition, wherein the second operand has at least one non-nullable column;

programmatically transforming the full outer join query into a union query of a left outer join of the first and second operands, and a right outer join of the first and second operands; and

filtering from the answer set all the matched rows of the right outer join.

10. The method of claim 9 wherein the step of adding a non-nullable column further comprises the step of including in the non-nullable column at least one of: a primary key; a row identifier; and a tuple identifier.

11. In a relational database management system utilizing a data processor for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of tuples having at least one column, both tables being tables other than base tables, wherein the data contained in the relational database management system is retrievable by means of query language queries to the database management system, a data processor implemented method for returning an answer set from the data in the tables responsive to a query to the database management system, the answer set requiring the matching of tuples from the first table and the second table, the method comprising the steps of:

for a full outer join condition, receiving a full outer join query of the tuples of the first operand and the tuples of the second operand, with a join condition, wherein the second operand has the property of being free of non-nullable columns;

programmatically transforming the full outer join query into a union query of a left outer join of the first and second operands and a right outer join of the first and second operands; and

filtering from the answer set all the matched rows of the right outer join.

15. In a relational database management system utilizing a data processor for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of tuples having at least one column, wherein the data contained in the relational database management system is retrievable by means of query language queries to the

database management system, a data processor implemented method for returning an answer set from the data in the tables responsive to a query to the database management system, the method comprising the steps of:

identifying the query as a full outer join query having an answer set including tuplesles of the first table that do not match tuples of the second table; and

executing an anti-join query of the first and second tables with a join condition using a run time operator with early out property.

16. The method of claim 15 wherein the join condition is from the set of comparison operators consisting of: IS NULL; LIKE; EQUAL; DOES NOT EQUAL; IS GREATER THAN; IS LESS THAN; IS GREATER THAN OR EQUAL TO; and IS LESS THAN OR EQUAL TO.

18. In a relational database management system utilizing a data processor for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of tuples having at least one column, wherein the data contained in the relational database management system is retrievable by means of query language queries to the database management system, a data processor implemented method for returning an answer set from the data in the tables responsive to a query to the database management system, the answer set comprising of the first table tuples that do not match the second table tuples, the method comprising the steps of:

receiving an anti-join query of the first and second tables with a join condition, wherein the second table has at least one non-nullable column;

programmatically transforming the anti-join into a left outer join of the first and second tables with an early out property; and

filtering from the answer set all the matched rows of the left outer join.

20. The method of claim 19 wherein the step of adding a non-nullable column further comprises the step of including in the non-nullable column at least one of: a primary key; a row identifier; and a tuple identifier.

21. In a relational database management system utilizing a data processor for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of tuples having at least one column, wherein the data contained in the relational database management system is retrievable by means of query language queries to the database management system, a data processor implemented method for returning an answer set from the data in the tables responsive to a query to the database management system, the answer set comprising tuples of the first table that do not match the tuples of the second table, the method comprising the steps of:

receiving an anti-join query of the first and second tables with a join condition, wherein the second table has the property of being free of non-nullable columns;

programmatically transforming the anti-join into a left outer join with an early out property; and

filtering from the answer set all the matched rows of the left outer join.

24. In a relational database management system utilizing a data processor for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of tuples having at least one column, wherein the data contained in the relational database management system is retrievable by means of query language queries to the database management system, a data processor implemented method for returning an

answer set from the data in the tables responsive to a query to the database management system, the answer set requiring the matching of tuples from the first table and the second table, the method comprising the steps of:

receiving a full outer-join query of the tuples of the first tables with a join condition;

programmatically transforming the full outer join query into a union query of a left outer join of the first and second operands, and an anti-join of the first and second operands using a new run time operator.

25. The method of claim 24 wherein the join condition is from the set of comparison operators consisting of: IS NULL; LIKE; EQUALS; DOES NOT EQUAL; IS GREATER THAN; IS LESS THAN; IS GREATER THAN OR EQUAL TO; and IS LESS THAN OR EQUAL TO.

27. A computer-implemented apparatus, with bus means, processor means, data storage means, input and output means and a relational database management system, for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of tuples having one or more columns, wherein the data contained in the relational database management system is retrievable by means of query language queries to the database management system, which apparatus returns an answer set from the data in the tables responsive to a query to the database management system, the apparatus comprising programmatic means for identifying the query as a full outer join query having an answer set requiring the matching of tuples from the first table and the second table, and programmatic means for joining the tuples of the first and second tables using a full outer join with a join condition, the join condition having the property of being free of restrictions as to the type of join condition specified, the means for joining comprising programmatic means for unioning a left outer join with a right outer join less the matched tuples from the right outer join.

28. The apparatus of claim 27 wherein the join condition is from the set of comparison operators consisting of: IS NULL; LIKE; EQUALS; DOES NOT EQUAL; IS GREATER THAN; IS LESS THAN; IS GREATER THAN OR EQUAL TO; and IS LESS THAN OR EQUAL TO.

30. A computer-implemented apparatus, with bus means, processor means, data storage means, input and output means and a relational database management system, for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of tuples having at least one column, wherein the data contained in the relational database management system is retrievable by means of query language queries to the database management system, the apparatus for returning an answer set from the data in the tables responsive to a query to the database management system, the apparatus comprising programmatic means for identifying the query as a full outer join query having an answer set requiring the matching of tuples from the first table and the second table, and programmatic means for joining the tuples of the first and second tables using a full outer join with any join condition specifiable in the query language, the means for joining comprising means for unioning a left outer join with a right outer join less the matched tuples from the right outer join.

31. The apparatus of claim 30 wherein the join condition is from the set of comparison operators consisting of: IS NULL; LIKE; EQUALS; DOES NOT EQUAL; IS GREATER THAN; IS LESS THAN; IS GREATER THAN OR EQUAL TO; and IS LESS THAN OR EQUAL TO.

33. In a relational database management system utilizing a data processor for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of

tuples having at least one column, wherein the data contained in the relational database management system is retrievable by means of query language queries to the database management system, a data processing apparatus ("apparatus") for returning an answer set from the data in the tables responsive to a query to the database management system, the answer set requiring the matching of tuples from the first table and the second table, the apparatus comprising:

for a full outer join condition, programmatic means for receiving a full outer join query of the tuples of the first operand and the tuples of the second operand, with a join condition, wherein the second operand has at least one non-nullable column;

programmatic means for transforming the full outer join query into a union query of a left outer join of the first and second operands and a right outer join of the first and second operands; and

programmatic means for filtering from the answer set all the matched rows of the right outer join.

36. The apparatus of claim 35 wherein the programmatic means for adding a non-nullable column further comprises programmatic means for including in the non-nullable column at least one of: a primary key; a row identifier; and a tuple identifier.

37. In a relational database management system utilizing a data processor for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of tuples having at least one column, wherein the data contained in the relational database management system is retrievable by means of query language queries to the database management system, a data processing apparatus ("apparatus") for returning an answer set from the data in the tables responsive to a query to the database management system, the answer set requiring the matching of tuples from the first table and the second table, the apparatus comprising:

for a full outer join condition, programmatic means for receiving a full outer join query of the tuples of the first operand and the tuples of the second operand, with a join condition, wherein the second operand has the property of being free of non-nullable columns;

programmatic means for transforming the full outer join query into a union query of a left outer join of the first and second operands and a right outer join of the first and second operands; and

programmatic means for filtering from the answer set all the matched rows of the right outer join.

41. In a relational database management system utilizing a data processor for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of tuples having at least one column, wherein the data contained in the relational database management system is retrievable by means of query language queries to the database management system, a data processor for returning an answer set form the data in the tables responsive to a query to the database management system, the data processor comprising:

means for identifying the query as a full outer join query having an answer set including tuples of the first table that do not match tuples of the second table; and

means for executing an anti-join query of the first and second tables with a join condition using a new run time operator with early out property.

42. The data processor of claim 41 wherein the join condition is from the set of comparison operators consisting of: IS NULL; LIKE; EQUAL; DOES NOT EQUAL; IS GREATER THAN; IS LESS THAN; IS GREATER THAN OR EQUAL TO; and IS LESS THAN OR EQUAL TO.

44. In a relational database management system utilizing a data processor for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of tuples having at least one column, wherein the data contained in the relational database management system is retrievable by means of query language queries to the database management system, a data processor for returning an answer set from the data in the tables responsive to a query to the database management system, the answer set comprising of the first table tuples that do not match the second table tuples, the data processor comprising:

means for receiving an anti-join query of the first and second tables with a join condition, wherein the second table has at least one non-nullable column;

means for programmatically transforming the anti-join into a left outer join of the first and second tables with an early out property; and

means for filtering from the answer set all the matched rows of the left outer join.

46. The data processor of claim 45 wherein the function of adding a non-nullable column further comprises the function of including in the non-nullable column at least one of: a primary key; a row identifier; and a tuple identifier.

47. In a relational database management system utilizing a data processor for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of tuples having at least one column, wherein the data contained in the relational database management system is retrievable by means of query language queries to the database management system, a data processor for returning an answer set from the data in the tables responsive to a query to the database management system, the answer set comprising of the first table tuples that do not match the second table tuples, the data processor comprising:

means for receiving an anti-join query of the first and second tables with a join condition, wherein the second table has the property of being free of non-nullable columns;

means for programmatically transforming the anti-join into a left outer join with an early out property; and

means for filtering from the answer set all the matched rows of the left outer join.

50. In a relational database management system utilizing a data processor for storing data in the form of at least a first table defining a first operand and a second table defining a second operand, the first and second tables comprised of tuples having at least one column, wherein the data contained in the relational database management system is retrievable by means of query language queries to the database management system, a data processor for returning an answer set from the data in the tables responsive to a query to the database management system, the answer set requiring the matching of tuples from the first table and the second table, the data processor comprising:

means for receiving a full outer-join query of the tuples of the first tables with a join condition; and

means for programmatically transforming the full outer join query into a union query of a left outer join of the first and second operands, and an anti-join of the first and second operands using a run time operator.

51. The data processor of claim 50 wherein the join condition is from the set of comparison operators consisting of: IS NULL; LIKE; EQUALS; DOES NOT EQUAL; IS GREATER THAN; IS LESS THAN; IS GREATER THAN OR EQUAL TO; and IS LESS THAN OR EQUAL TO.

Previous Doc Next Doc Go to Doc#

# Print Request Result(s)

**Printer Name: ran_3c18_gbrcptr**
**Printer Location: ran__3c18**
**Number of Copies Printed : 1**

- US20020111949: Ok
- US006801914: Ok
- US006665640: Ok
- US005367675: Ok

OK Back to List

# Freeform Search

| | |
|---|---|
| Database: | US Pre-Grant Publication Full-Text Database<br>US Patents Full-Text Database<br>US OCR Full-Text Database<br>EPO Abstracts Database<br>JPO Abstracts Database<br>Derwent World Patents Index<br>IBM Technical Disclosure Bulletins |
| Term: | L8 and (rewrit$ near sql) |
| Display: | 50 Documents in Display Format: - Starting with Number 1 |
| Generate: | ○ Hit List [✓ Hit Count] ○ Side by Side ○ Image |

Search Clear Interrupt

## Search History

**DATE: Friday, July 08, 2005** Printable Copy Create Case

| Set Name side by side | Query | Hit Count | Set Name result set |
|---|---|---|---|
| *DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR* | | | |
| L9 | L8 and (rewrit$ near sql) | 6 | L9 |
| L8 | L7 and (master near table) | 239 | L8 |
| L7 | 707/$.ccls. | 27808 | L7 |
| L6 | l1 and (revis$ near sql) | 3 | L6 |
| L5 | L4 and identifier | 4 | L5 |
| L4 | L1 and (join near condition) | 7 | L4 |
| L3 | L2 and (master near table) | 1 | L3 |
| L2 | L1 and (join near table) | 10 | L2 |
| L1 | rewrit$ near sql | 53 | L1 |

END OF SEARCH HISTORY

# Freeform Search

**Database:**
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

**Term:** `L8 and (rewrit$ near sql)`

**Display:** 50 **Documents in Display Format:** - **Starting with Number** 1

**Generate:** Hit List [✓ Hit Count] Side by Side Image

Search Clear Interrupt

**Search History**

**DATE: Friday, July 08, 2005** Printable Copy Create Case

| **Set Name** side by side | **Query** | **Hit Count** | **Set Name** result set |
|---|---|---|---|
| *DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR* | | | |
| L9 | L8 and (rewrit$ near sql) | 6 | L9 |
| L8 | L7 and (master near table) | 239 | L8 |
| L7 | 707/$.ccls. | 27808 | L7 |
| L6 | l1 and (revis$ near sql) | 3 | L6 |
| L5 | L4 and identifier | 4 | L5 |
| L4 | L1 and (join near condition) | 7 | L4 |
| L3 | L2 and (master near table) | 1 | L3 |
| L2 | L1 and (join near table) | 10 | L2 |
| L1 | rewrit$ near sql | 53 | L1 |

END OF SEARCH HISTORY

L9: Entry 6 of 6 File: USPT Nov 22, 1994

DOCUMENT-IDENTIFIER: US 5367675 A
**** See image for Certificate of Correction ****
TITLE: Computer automated system and method for optimizing the processing of a query in a relational database system by merging subqueries with the query

Brief Summary Text (9):
Subqueries are often used in SQL update, delete and insert statements. For example, to delete all the accounts from the MASTER table which have `D` entries in the REVISION table, one may, write:

Brief Summary Text (13):
In Example 2, it is necessary to know which tuples in the MASTER table have been marked for deletion in the REVISION table. A commonly used method of processing queries containing a subquery is to evaluate the subquery for each row in the outer query. In Example 2 this approach would result in evaluating the subquery for every row in the MASTER table which could be quite large.

Brief Summary Text (17):
2. The outer table is very large, the inner table of the subquery after restriction is very small, and only a few rows in the outer table satisfy the subquery. In Example 2, the MASTER table is very large but only a few rows are to be deleted. Scanning every row in a large table in order to delete a few rows that satisfy the subquery is an inefficient way of subquery processing.

Brief Summary Text (43):
As mentioned above, this method can also apply to the optimization of table expressions. Users cannot always rewrite SQL subqueries (or views or table expressions) using the join operator due to the semantic differences between join and subquery. The DBMS can perform such a transformation by special handling of the semantic differences (such as duplicates, empty table, etc.) between join and subquery as this invention describes. Methods are disclosed for efficient execution of subqueries, views and table expressions by merging (composing) them into queries wherever possible, hence taking advantage of better join orders, and eliminating correlation.

Current US Original Classification (1):
707/2

L9: Entry 5 of 6 File: USPT Dec 16, 2003

DOCUMENT-IDENTIFIER: US 6665640 B1
TITLE: Interactive speech based learning/training system formulating search queries based on natural language parsing of recognized user queries

Detailed Description Text (128):
In a preferred embodiment, an independent database is used for each Course. Each database in turn can include three types of tables as follows: a Master Table as illustrated in FIG. 7A, at least one Chapter Table as illustrated in FIG. 7B and at least one Section Table as illustrated in FIG. 7C.

Detailed Description Text (129):
As illustrated in FIG. 7A, a preferred embodiment of a Master Table has six columns--Field Name 701A, Data Type 702A, Size 703A, Null 704A, Primary Key 705A and Indexed 706A. These parameters are well-known in the art of database design and structure. The Master Table has only two fields--Chapter Name 707A and Section Name 708A. Both ChapterName and Section Name are commonly indexed.

Detailed Description Text (130):
A preferred embodiment of a Chapter Table is illustrated in FIG. 7B. As with the Master Table, the Chapter Table has six (6) columns--Field Name 720, Data Type 721, Size 722, Null 723, Primary Key 724 and Indexed 725. There are nine (9) rows of data however, in this case,--Chapter_ID 726, Answer_ID 727, Section Name 728, Answer_Tide 729, PairedQuestion 730, AnswerPath 731, Creator 732, Date of Creation 733 and Date of Modification 734.

Detailed Description Text (132):
A preferred embodiment of a Section Table is illustrated in FIG. 7D. The Section Table has six (6) columns--Field Name 740, Data Type 741, Size 742, Null 743, Primary Key 744 and Indexed 745. There are seven (7) rows of data--Answer_ID 746, Answer_Title 747, PairedQuestion 748, AnswerPath 749, Creator 750, Date of Creation 751 and Date of Modification 752. These names correspond to the same fields, columns already described above for the Master Table and Chapter Table.

Detailed Description Text (137):
As illustrated in FIG. 10, a Full-Text Query Process is implemented as follows: 1. A query 1001 that uses a SQL full-text construct generated by DB processor 186 is submitted to SQL Relational Engine 1002. 2. Queries containing either a CONTAINS or FREETEXT predicate are rewritten by routine 1003 so that a responsive rowset returned later from Full-Text Provider 1007 will be automatically joined to the table that the predicate is acting upon. This rewrite is a mechanism used to ensure that these predicates are a seamless extension to a traditional SQL Server. After the compiled query is internally rewritten and checked for correctness in item 1003, the query is passed to RUN TIME module 1004. The function of module 1004 is to convert the rewritten SQL construct to a validated run-time process before it is sent to the Full-Text Provider, 1007. 3. After this, Full-Text Provider 1007 is invoked, passing the following information for the query: a. A ft_search_condition parameter (this is a logical flag indicating a full text search condition) b. A name of a full-text catalog where a full-text index of a table resides c. A locale ID to be used for language (for example, word breaking) d. Identities of a database, table, and column to be used in the query e. If the query is comprised of

more than one full-text construct; when this is the case Full-text provider 1007 is invoked separately for each construct. 4. SQL Relational Engine 1002 does not examine the contents of ft_search_condition. Instead, this information is passed along to Full-text provider 1007, which verifies the validity of the query and then creates an appropriate internal representation of the full-text search condition. 5. The query request/command 1008 is then passed to Querying Support 1011A. 6. Querying Support 1012 returns a rowset 1009 from Full-Text Catalog 1013 that contains unique key column values for any rows that match the full-text search criteria. A rank value also is returned for each row. 7. The rowset of key column values 1009 is passed to SQL Relational Engine 1002. If processing of the query implicates either a CONTAINSTABLE( ) or FREETEXTTABLE( ) function, RANK values are returned; otherwise, any rank value is filtered out. 8. The rowset values 1009 are plugged into the initial query with values obtained from relational database 1006, and a result set 1015 is then returned for further processing to yield a response to the user.

Current US Cross Reference Classification (3):
707/4

Previous Doc Next Doc Go to Doc#

First Hit Previous Doc Next Doc Go to Doc#

☑ Generate Collection Print

L9: Entry 3 of 6 File: PGPB Aug 15, 2002

DOCUMENT-IDENTIFIER: US 20020111949 A1
TITLE: Persistent client-server database sessions

Current US Classification, US Primary Class/Subclass:
707/10

Detail Description Paragraph:
[0073] Consider a database session involving a data analysis query, similar to those in the TPC-D benchmark. This query involves three database tables: a master customer table 42(1), a detail orders table 42(2), and an invoice table 42(3). The task is to extract appropriate records for a customer with the last name "Smith" from the customer table, find that customer's current orders from the detail orders table, and aggregate the order totals into an invoice table. This client application might be coded as follows.

Detail Description Paragraph:
[0097] At step 264 in FIG. 6a, once the server returns a response indicating the rewritten SQL statement has successfully executed, the driver manager 38 reads the table description metadata and reformats it into a CREATE TABLE statement. The driver manager 38 sends the CREATE TABLE statement to the database server 24 (step 266 in FIG. 6b). In response, the database server 24 creates an empty persistent table 44(1) in stable database 28 to hold an eventual result set obtained from the "Smith" query (step 268 in FIG. 6b).

Previous Doc Next Doc Go to Doc#

L9: Entry 4 of 6 File: USPT Oct 5, 2004

DOCUMENT-IDENTIFIER: US 6801914 B2
TITLE: Persistent client-server database sessions

Detailed Description Text (48):
Consider a database session involving a data analysis query, similar to those in the TPC-D benchmark. This query involves three database tables: a master customer table 42(1), a detail orders table 42(2), and an invoice table 42(3). The task is to extract appropriate records for a customer with the last name "Smith" from the customer table, find that customer's current orders from the detail orders table, and aggregate the order totals into an invoice table. This client application might be coded as follows.

Detailed Description Text (68):
At step 264 in FIG. 6a, once the server returns a response indicating the rewritten SQL statement has successfully executed, the driver manager 38 reads the table description metadata and reformats it into a CREATE TABLE statement. The driver manager 38 sends the CREATE TABLE statement to the database server 24 (step 266 in FIG. 6b). In response, the database server 24 creates an empty persistent table 44 (1) in stable database 28 to hold an eventual result set obtained from the "Smith" query (step 268 in FIG. 6b).

Current US Original Classification (1):
707/10

Current US Cross Reference Classification (1):
707/100

Current US Cross Reference Classification (2):
707/200

L5: Entry 3 of 4 File: PGPB Jul 18, 2002

DOCUMENT-IDENTIFIER: US 20020095405 A1
TITLE: View definition with mask for cell-level data access control

Detail Description Paragraph:
[0096] Referring to FIGS. 14a -14c, and 15, we show some restrictions for selection conditions on mask columns, since it may break a mask; i.e. compromise the mask feature thereby permitting unauthorized access to secured data. There are two kinds of selection conditions to consider: One is a condition on a single relation. The other selection condition which can break a mask is a select condition between multiple relations. We call these conditions a "selection condition" and a "join condition," respectively.

Detail Description Paragraph:
[0108] Another way of breaking the mask is through the use of a join condition. A join condition is a kind of a selection condition. Consequently, it should be processed in the same way as other selection conditions. If we allow using a join condition without any restrictions, a user can break the mask in the following way: First, a temporary table TMP 1602 is created which includes the column P_NM. Assuming patient ERIS is the target, the user would enter the value `ERIS` into the P_NM column in the TMP table. The SQL statement in FIG. 14a is rewritten into the SQL statement 1610 shown in FIG. 16, which includes a join operation using the P_NM column as the join key. The result shown in FIG. 14c is thereby obtained because `ERIS` is common to the TMP table and to the INP_FACT view of FIG. 12. Therefore, further in accordance with the invention, if a selection condition includes a JOIN operation, the selection is restricted per PROCESS 2 during the rewrite.

Detail Description Paragraph:
[0109] However, there is sometimes a need to join a fact table with reference tables which involve mask columns. Referring to FIG. 17, for example, assume that there is an MD_FACT view 1702 that has the following columns: MD_ID, D_NM, and DEPT. FIG. 17 shows the MD_FACT view and its view definition 1704. Since the column DEPT is not a mask column, a user should be able to get a total cost grouped by each department. FIG. 18a and FIG. 18b show the SQL statements that a user wants to execute and the expected result. However, if we restrict join conditions between INPT_FACT and MD_FACT, a user MD_ID=3333 will get a different result shown in FIG. 18c.

Detail Description Paragraph:
[0110] To solve this issue, a view with mask according to the invention includes a join_prmt clause (1113, FIG. 11) so that a user can declare a join permission for mask columns. A join condition is permitted if all mask columns in the join condition expression have join permission with all other columns in the expression, except between the columns in the same view or table. For example, FIG. 19 shows the view with mask definitions for INPT_FACT 1902 and MD_FACT 1904 having join permission clauses 1913a and 1913b, respectively. These clauses permit the execution of a JOIN operation between MD_FACT and INPT_FACT using MD_ID as a join key.

Detail Description Paragraph:
[0143] In accordance with the present invention, a notation for external column

references is introduced to solve this problem. FIG. 26 shows view definitions for the views in FIG. 25. The mask condition predicate REF(PT_ID).MD_ID=user_id uses an external column reference. REF(PT_ID) is replaced by a table or view that have a join condition with a join key, PT_ID. FIG. 27 shows a syntax diagram of an external column reference. An external table or view is identified by the REF( ) expression 2702 with a set of parameters 2704 that represent join keys. The column followed by the REF( ) expression represents a column that the mask condition refers.

Detail Description Paragraph:
[0148] FIG. 28 shows the case where the unification of an external column reference succeeds. The b.P_NM in the original SQL statement will be translated into a CASE expression that includes REF(b.PT_ID).MD_ID. The external reference identifier REF (b.PT_ID) searches the join condition that has a join key, b.PT_ID, and it finds a target relation a. Consequently, REF(b.PT_ID).MD_ID is replaced with a.MD_ID.

Detail Description Paragraph:
[0149] Two cases where the unification fails are shown in FIG. 29 and FIG. 30. In the first case in FIG. 29, since the original query does not have any join condition, there is no matching relation for REF(PT_ID). Therefore, the predicate term, REF(PT_ID).MD_ID=user_id, is replaced by FALSE, and then the execution result of the translated SQL statement does not have any visible P_NM. In the second case in FIG. 30, the external column reference expression have two unification candidates, b and C. Therefore, the predicate term is replaced by FALSE and therefore no P_NM value will be returned as a result.

Detail Description Paragraph:
[0155] In order to apply the first solution, we need to expand all subqueries to check selection conditions on mask columns and aggregation functions. However, it is difficult for end-users to predict the expansion result of the query, therefore, users sometimes may have a different result from what they expected. Therefore, we choose the second solution to keep the semantics as simple as possible. For example, the mask for INPT_FACT.PT_ID is applied for the subquery in the query in FIG. 32a, therefore the subquery returns the following PT_ID values: PT_ID=12345 for the first row of INPT_FACT, and PT_ID=NULL for the second and third row of INPT_FACT. (The fourth row will be filtered by the condition DRG BETWEEN 120 AND 129.) Since the IN predicate in the main SELECT clause is recognized as a join condition between PT_FACT and INPT_FACT using PT_ID as a key, the query returns the result in FIG. 32b.

CLAIMS:

15. The method of claim 13 wherein said providing includes receiving a view definition having a mask condition, said mask condition includes one or more column identifiers and a mask predicate, said one or more column identifiers being associated with said one or more mask values, information that is categorized under said one or more column identifiers being referred to as masked information, wherein if said mask predicate evaluates to a first logic value then masked information contained in said first result appears in said final result; wherein if said mask predicate evaluates to a second logic value then said replacing includes replacing masked information contained in said result first result with one of said one or more mask values.

Previous Doc Next Doc Go to Doc#